Supplementary Material for Multi-Modal Fusion Transformer for End-to-End Autonomous Driving

Aditya Prakash[*1] Kashyap Chitta[*1,2] Andreas Geiger[1,2]
[1]Max Planck Institute for Intelligent Systems, Tübingen [2]University of Tübingen
{firstname.lastname}@tue.mpg.de

Abstract

*In this **supplementary document**, we first provide additional information regarding our architecture (Sec. 1). We then describe our data generation pipeline and dataset statistics (Sec. 2). Further, we discuss details regarding the baselines (Sec. 3) and the hyper-parameter choices for all the models (Sec. 4). Finally, we present additional results and analysis to supplement our findings in the main paper (Sec. 5). The **supplementary video** contains qualitative comparisons of our TransFuser model against Geometric Fusion on the evaluation routes.*

1. Architecture Details

In this section, we provide additional details regarding the multi-scale feature fusion component of the TransFuser and the low-level PID controller that converts waypoints to steer, throttle, and brake of the vehicle.

1.1. Multi-Scale Feature Fusion

The TransFuser takes in an RGB image and LiDAR BEV representation as inputs and processes them with ResNet-34 [8] and ResNet-18 [8] modules respectively. It uses several transformers for the fusion of intermediate feature maps between both modalities. For an input of resolution 256×256 pixels, the ResNet module produces intermediate feature maps of dimensions $64 \times 64 \times 64$, $32 \times 32 \times 128$, $16 \times 16 \times 256$ and $8 \times 8 \times 512$ after max pool operations in different ResNet blocks. Since processing feature maps at high spatial resolutions is computationally expensive, we downsample higher resolution feature maps from the early encoder blocks using average pooling to a fixed resolution of 8×8 before passing them as inputs to the fusion transformer. This results in feature maps of dimensions $8 \times 8 \times 64$, $8 \times 8 \times 128$, $8 \times 8 \times 256$ and $8 \times 8 \times 512$. The feature extractor branches of both the modalities provide feature maps with the dimensions described earlier. Since we set $D_f = 512$ for each transformer, we use 1×1 convolutions to match the embedding dimension of each feature map to 512.

Next, we describe the fusion mechanism between the RGB and LiDAR BEV feature maps of dimension $8 \times 8 \times 512$. The RGB and LiDAR BEV feature maps of dimension $8 \times 8 \times 512$ are stacked together to form a tensor of dimension $(2 * 8 * 8) \times 512$, where 2 represents RGB and LiDAR BEV modalities. This tensor is passed as input to the transformer which outputs a feature of dimension $(2 * 8 * 8) \times 512$. It is then reshaped into 2 tensors of dimension $8 \times 8 \times 512$ each for the 2 modalities. Each of these 2 tensors is then upsampled using bilinear interpolation to match the original feature resolution and processed through 1×1 convolutions to match the embedding dimension of the original feature maps. It is then combined with the existing feature maps via element-wise summation. We pass the fused features back into the feature extraction branches since this provides the deeper ResNet layers access to the global context of the 3D scene encoded in the earlier layers. This results in a 512-dimensional feature vector output from both the image and LiDAR BEV stream, which is combined via element-wise summation. This 512-dimensional feature vector constitutes a compact representation of the environment which incorporates global contextual reasoning.

[*]indicates equal contribution

Algorithm 1: Generating an action from waypoints.

input : $v, \{\mathbf{w}_t\}_{t=1}^{T}$; // velocity and waypoints
output: steer $\in (-1, 1)$, throttle $\in (0, 1)$, brake $\in (0, 1)$; // vehicle control
$\gamma = 0$
for $t \leftarrow 1$ **to** $T - 1$ **do**
  γ += $\lambda_t ||\mathbf{w}_{t+1} - \mathbf{w}_t||$; // desired velocity
$\omega = \frac{(\mathbf{w}_1 + \mathbf{w}_2)}{2}$
$\alpha = \tan^{-1}\left(\frac{\omega[1]}{\omega[0]}\right)$; // aim direction
steer = LatPID(α)
if $\gamma < \beta_{min}$ **or** $\gamma < v\beta_{ratio}$ **then**
  throttle $= 0$
  brake $= 1$
else
  throttle = LonPID($\gamma - v$)
  brake $= 0$

1.2. PID Controller

To convert waypoints to vehicle controls, we use two PID controllers. This process is summarized in Algorithm 1. We use the same configuration as in the author-provided codebase of [2]. We first compute the $T - 1$ vectors between waypoints of consecutive time-steps, with $T = 4$ in our configuration. The longitudinal controller (LonPID in Algorithm 1) takes in a weighted average of these vectors and tries to match the vehicle velocity v to the desired velocity γ as much as possible. We find the weights $\lambda = \{1, 0, 0\}$, for the $T - 1$ vectors, prioritizing the closest pair of waypoints, give the best empirical results. The lateral PID controller (LatPID in Algorithm 1) orients the vehicle along the aim direction α which is computed as the orientation of the midpoint of $\mathbf{w}_1$ and $\mathbf{w}_2$. Each controller has 3 gain parameters, which are tuned on a subset of our training routes. For the longitudinal controller, we set $K_p = 5.0, K_i = 0.5, K_d = 1.0$ and for the lateral controller, we set $K_p = 1.25, K_i = 0.75, K_d = 0.3$. Both controllers use a buffer of size 40 to approximate the integral term as a running average. The other parameters used are a brake threshold speed β_{min} and brake speed ratio β_{ratio}, which we tune jointly with the gain parameters, and set to 0.4 and 1.1 respectively.

2. Datasets

In this section, we describe our data generation pipeline with details about the sensor configuration, expert, routes, and scenarios used in the CARLA simulator. We additionally provide statistics regarding the distribution of navigational commands, weather conditions, and dynamic agents in our dataset.

2.1. Sensor Configuration

We use the front camera with a field of view (FOV) of 100° for the RGB image input. The camera is mounted at a height of 2.3m from the ground level of the ego-vehicle and located 1.3m in front of the centroid of the ego-vehicle. We offset the cameras forward by 1.3m to avoid the hood of the ego-vehicle from occluding a portion of the rendered images. We extract images from this camera at a resolution of 400 × 300 pixels. For the LiDAR point cloud, we use a ray-cast-based Velodyne 64 LiDAR with an 85m range and rotation frequency of 10 FPS. The upper FOV of the LiDAR is set at 10° and the lower FOV is set at -30°. The LiDAR sensor is mounted at a height of 2.5m from the ground level of the ego-vehicle and located 1.3m in front of the ego-vehicle. We further use additional sensors such as IMU to get the orientation, GPS for localization, and speedometer to get the current speed of the ego-vehicle.

2.2. Expert

For generating training data, we roll out an expert policy designed to drive using privileged information from the simulator. The expert policy consists of an A* planner followed by 2 PID controllers (for lateral and longitudinal control). Several heuristics based on the global position of the dynamic agents and traffic lights are used by the expert to avoid collisions and

traffic violations. We build upon the code[1] provided by the authors of [2]. We make two modifications to their code, (1) We change the distance and speed heuristics used for avoiding collisions and following traffic lights to make the expert more cautious, (2) We add stop sign avoidance functionality since it is penalized in the CARLA Leaderboard.

Given the route to be followed as a sequence of sparse waypoints, the expert policy uses the A* planner to interpolate these sparse waypoints to dense waypoints with a sampling resolution of 1m. From these dense waypoints, it selects two waypoints - the nearest waypoint after 4m from its current position as the near node and the nearest waypoint after 7m from its current position as the far node. It uses the orientation of the near node as the aim direction, which is fed to the lateral PID controller, with the steering value as its output. It uses the orientation of the far node as an indication of an approaching turn. The expert uses a discrete set of speed values, $\{0, 4, 7\}$m/s as the target speed for driving, depending on its current circumstances. If the expert is in close proximity to pedestrians, vehicles, red light, or a stop sign, then the target speed is set to 0 m/s. If the expert has to steer by more than 5° or is approaching a turn, as indicated by the orientation of the far node, then the target speed is set to 4 m/s. Under every other circumstance, the target speed is set to 7 m/s. The longitudinal controller then tries to match the speed of the expert policy to this target speed as much as possible. For the longitudinal controller, we set $K_p = 5.0, K_i = 0.5, K_d = 1.0$ and for the lateral controller, we set $K_p = 1.25, K_i = 0.75, K_d = 0.3$. Both controllers use a buffer of size 40 to approximate the integral term as a running average.

2.3. Routes

The CARLA Leaderboard repository provides a set of 76 routes as a starting point for training (50 routes) and evaluating (26 routes) the agent[2]. These routes were originally released along with the 2019 CARLA challenge. These routes span 6 towns and each of them is defined by a sequence of waypoints. In addition to these, we also generate extra routes to incorporate more turnings and intersections in the training data using the procedure described below.

We first sample the intersections present in the town map. For this, we use the positions of the traffic lights since most intersections contain traffic lights. For each such sampled intersection, we generate multiple pairs of start and end waypoints, by sampling points along the canonical axes in a 100m $\times$ 100m grid centered at the intersection since road segments at intersections in CARLA are aligned with the canonical axes in world coordinates. From this set of sampled points, we select pairs of points that lie on different road segments of the intersections. In this manner, we sample 8 pairs of turn routes at each intersection. However, if any of these sampled points do not lie on a road segment, e.g. in 3-way intersections, the simulator automatically maps the point lying off the road segment to the nearest point on a road segment. This results in several long routes encompassing multiple intersections. Next, we explicitly sample individual turns along these routes. For this, we interpolate each of the routes, obtaining a dense set of waypoints along each. Then, we sample points along these routes where the navigational command changes from 'Follow Lane' to 'Turn Left', 'Turn Right' or 'Go Straight'. These points indicate the starting position of an intersection. When the navigation command changes back to 'Follow Lane' along the interpolated route, that indicates the end of an intersection. In this manner, we sample several pairs of start and end points along each long route with each pair representing a single intersection of length 25-50m.

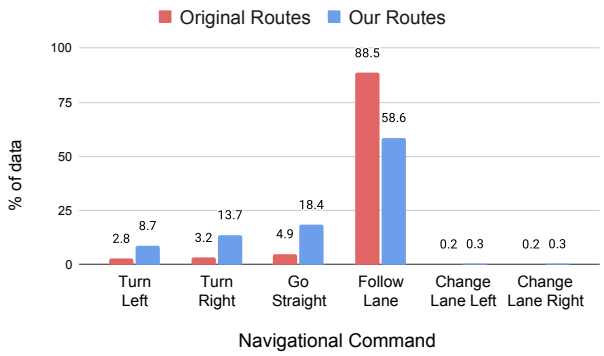
Adding these routes significantly improves the distribution of navigational commands in the training dataset (Fig. 1a) and explicitly incorporates more turns and intersections. The total data collected using the original 76 routes contains around 70k frames whereas the data collected using our routes contains around 150k frames.

2.4. Scenarios

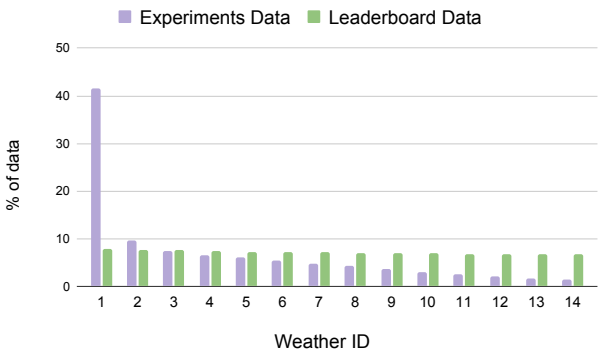
The CARLA Leaderboard repository also provides a set of scenarios[3] for each town. Each scenario is defined by (1) a trigger 'transform' which indicates the spawn location and the orientation of that scenario in a particular town and (2) 'other actors' which contains information about additional agents present in that scenario. However, this provided set contains only 3 scenarios - control loss without previous action, obstacle avoidance without prior action, and obstacle avoidance with prior action, whereas the Leaderboard evaluation consists of 10 scenarios, including situations such as other vehicles running red lights and unprotected turns. Therefore, we also include additional scenarios while generating training data, so that our agent is robust to these adversarial situations.

Specifically, we include Scenarios[4] 1, 3, 4, 7, 8, 9, 10 in our training dataset. Since we focus on safety-critical scenarios, e.g. pedestrians emerging from occluded regions to cross the road at random locations, vehicles running red lights, and unprotected turnings, we do not include Scenarios 5 and 6, which involve lane changing. We also do not include Scenario 2

[1] https://github.com/bradyz/2020_CARLA_challenge
[2] https://github.com/carla-simulator/leaderboard/tree/stable/data
[3] https://github.com/carla-simulator/leaderboard/blob/stable/data/all_towns_traffic_scenarios_public.json
[4] https://leaderboard.carla.org/scenarios/



(a) **Distribution of Navigational Commands.** We report the distribution of navigational commands for the original routes provided in the Leaderboard repository and our routes used for generating data. Our routes incorporate more turns and intersections in the training data.



(b) **Distribution of Weathers.** 1: ClearNoon, 2: ClearSunset, 3: CloudyNoon, 4: CloudySunset, 5: WetNoon, 6: WetSunset, 7: MidRainyNoon, 8: MidRainSunset, 9: WetCloudyNoon, 10: WetCloudySunset, 11: HardRainNoon, 12: HardRainSunset, 13: SoftRainNoon, 14: SoftRainSunset

since it occurs naturally during the driving in the presence of other dynamic agents. To generate trigger transforms for these additional scenarios, we consider the trigger transforms provided in the Leaderboard set of scenarios and sample additional spawn locations at a 5m distance along the canonical axes in a 10m $\times$ 10m grid centered at each available trigger position. For the spawn orientation, we select yaw $\in \{0°, 90°, 180°, 270°\}$. Therefore, for each provided trigger transform in a particular town, we generate 36 trigger transforms for each scenario. While the CARLA leaderboard framework[5] considers prioritized sampling of scenarios (scenarios with higher ID are given preference, e.g. scenario 10 is prioritized over scenario 9 and so on) in case multiple scenarios can be spawned at a particular trigger location, we consider uniform sampling (randomly selecting one scenario per trigger location) in our implementation.

2.5. Data Statistics

We use 7 towns (Fig. 2) and 14 weather conditions (Fig. 1b) for generating training data. We use the set of routes and scenarios generated by the process described in Sec. 2.3 and Sec. 2.4 and store data at 2 FPS. We sequentially vary the weather condition after every 30 seconds in each route. However, since a significant proportion of the routes used for generating data are short, the weather distribution is skewed towards the initial weathers (Experiments data in Fig. 1b) since all the 14 kinds of weather are not iterated through. This is not a problem for our internal evaluation since we fix the environmental conditions to 'ClearNoon' during evaluation, to decouple the main new challenge (scenarios) from factors related to visual generalization. However, for submission to the CARLA Leaderboard (Sec. 5), we regenerate our data while varying the weather condition after every 0.5 seconds of driving so that the distribution is uniform across all weathers (Leaderboard Data in Fig. 1b). The number of dynamic agents in each town is set as - Town01: 120, Town02: 100, Town03: 120, Town04: 200, Town05: 120, Town06: 150, Town07: 110, Town10: 120.

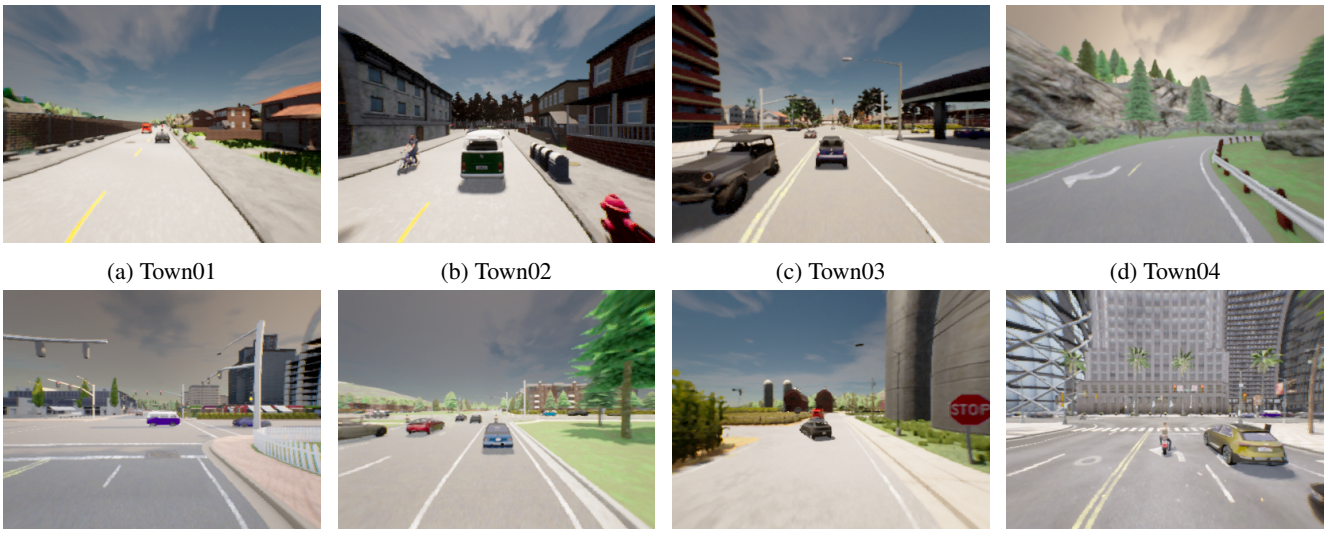
3. Baselines

In this section, we describe in detail the baselines (Tab. 1) used in our experiments - CILRS [5], LBC [2], Auto-regressive IMage based waypoint prediction (AIM), Late Fusion and Geometric Fusion.

3.1. CILRS

CILRS [5] is a conditional imitation learning method in which the agent learns to predict vehicle controls from a single front camera image while being conditioned on the navigational command. The front camera image is processed by a ResNet-34 encoder which outputs a 512-dimensional feature vector. In addition, CILRS also takes in the current speed $\hat{v}$ as input which is processed by a 1-layer MLP consisting of 256 units to produce a 512-dimensional feature vector, which is combined with the feature vector of the image branch via element-wise summation. While the original architecture combined the velocity vector and the image feature vector via concatenation followed by an MLP, we found element-wise summation

[5] https://github.com/carla-simulator/leaderboard



(a) Town01 (b) Town02 (c) Town03 (d) Town04

(e) Town05 (f) Town06 (g) Town07 (h) Town10

Figure 2: **CARLA Towns.** Town01, Town02, Town07 are EU style towns whereas the others are US style.

to work better empirically. This 512-dimensional vector is then fed to the controller module which is conditioned on discrete navigational commands. While the original architecture used 4 navigational commands (follow lane, turn left / turn right / go straight at intersection), we adapted it to include 2 additional commands for lane changes (left and right). Since the space of navigational commands is discrete, CILRS uses a conditional module to select one of several command branches based on the input command. The command branch directly outputs control values (steer, throttle, and brake).

Additionally, the output of the ResNet-34 image encoder in CILRS is used for predicting the current vehicle speed v, which is compared to the actual vehicle speed $\hat{v}$ with the velocity loss $\mathcal{L}_{\text{velocity}}$ (Eq. 1). The final loss function for CILRS is a weighted sum of an L_1 imitation loss (Eq. 1), between the predicted control $\mathbf{a}$ and the ground-truth expert control $\mathbf{a}^*$, and the velocity loss, with a scalar weight ζ. We set $\zeta = 0.05$ since it gives the best empirical performance.

$$\mathcal{L}_{\text{imitiation}} = ||\mathbf{a} - \mathbf{a}^*||_1 \qquad \mathcal{L}_{\text{velocity}} = ||v - \hat{v}||_1 \tag{1}$$

$$\mathcal{L} = \mathcal{L}_{\text{imitiation}} + \zeta \mathcal{L}_{\text{velocity}} \tag{2}$$

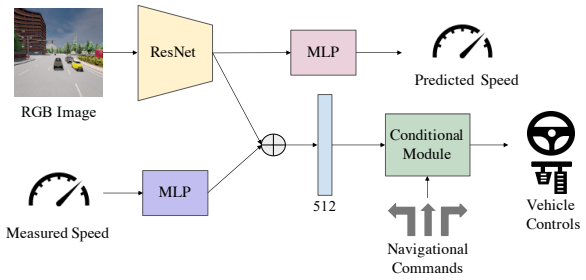
3.2. LBC

LBC is a knowledge distillation approach where a teacher model with access to ground truth BEV semantic maps is first trained using expert supervision to predict waypoints, followed by an image-based student model which is trained using supervision from the teacher. While the original LBC architecture uses only the front camera image as input and is conditioned on discrete navigational commands (similar to CILRS), the authors recently released an updated version of their architecture[6] with 2 major modifications, (1) multi-view camera inputs to a ResNet-50 encoder (front, 45° left, and 45° right images are stacked as channels), and (2) target heatmap as input (instead of the navigational command) which is formed by projecting the target point into front camera image coordinates. We directly use this codebase in our experiments, with one minor adaptation: the official implementation did not use the stop sign semantic class for the teacher model, which we include since this infraction is penalized during evaluation.

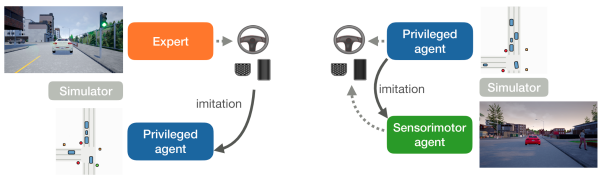
3.3. AIM

We implement our auto-regressive waypoint prediction network with an image-based ResNet-34 encoder, initialized from ImageNet [6] pre-trained weights, which takes just the front camera image as input. This baseline is equivalent to adapting the CILRS model to predict waypoints conditioned on sparse goal locations rather than vehicle controls conditioned on navigational commands. The image encoder used for this is the same as CILRS and our TransFuser model.

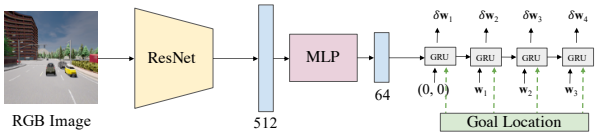
[6] https://github.com/bradyz/2020_CARLA_challenge



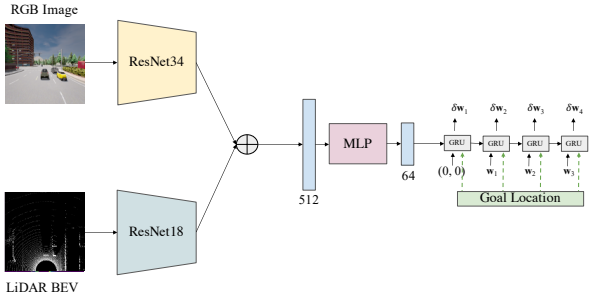
(a) CILRS



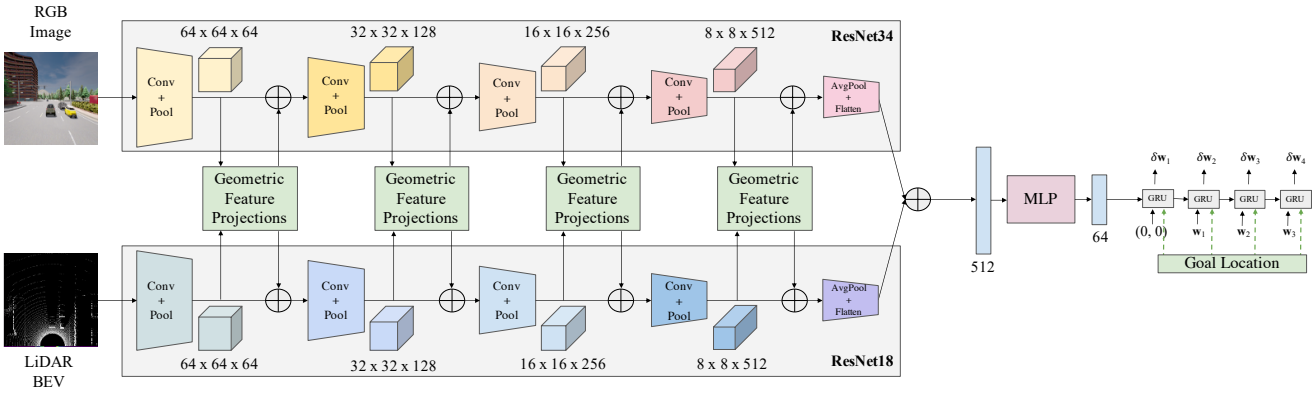
(b) LBC



(c) AIM



(d) Late Fusion



(e) Geometric Fusion

Table 1: **Baselines.** Illustration of baseline methods used in our experiments.

3.4. Late Fusion

We implement a version of our architecture where the RGB image and the LiDAR BEV streams are processed independently of each other. The image features are extracted using ResNet-34, which is initialized from ImageNet pre-trained weights, whereas the LiDAR BEV features are extracted using ResNet-18, which is trained from scratch. These features are then combined via element-wise summation and passed to the waypoint prediction network. This architecture is equivalent to removing the transformer modules from the TransFuser. This is similar to the encoder of [17] which also processed the image feature and LiDAR features separately. While Sobh et al. [17] used a learned controller module conditioned on discrete navigational commands (similar to CILRS), we use our auto-regressive waypoint prediction network since it gives a better empirical performance.

3.5. Geometric Fusion

We implement a multi-scale geometry-based fusion method, inspired by [11, 12], involving both image-to-LiDAR and LiDAR-to-image feature fusion. We unproject each 0.125m $\times$ 0.125m block in our LiDAR BEV representation into 3D space resulting in a 3D volume. We randomly select 5 points from the LiDAR point cloud lying in this 3D volume and

project them into the image space. We aggregate the image features of these points via element-wise summation before passing to a 3-layer MLP, consisting of 512 units each. The output of the MLP is then combined with the LiDAR BEV feature of the corresponding 0.125m $\times$ 0.125m block at multiple resolutions throughout the feature extractor.

Similarly, for each image pixel, we randomly select 5 points in the LiDAR point cloud which project to that pixel in image space, then project these points into the BEV space, aggregate their features via element-wise summation before passing to a 3-layer MLP, consisting of 512 units each. The output of the MLP is then combined with the features of the corresponding image pixels at multiple resolutions throughout the feature extractor.

Similar to the TransFuser, we downsample the higher spatial resolution feature maps to a fixed resolution of 8 $\times$ 8 and use 1 $\times$ 1 convolutions to match the embedding dimension of the feature maps to 512. After the features are combined, they are upsampled to the original resolution using bilinear interpolation and processed through 1 $\times$ 1 convolutions to match the embedding dimension of the original feature map, before being fed into the individual feature extractors of the two modalities. This baseline is equivalent to replacing the transformers in our architecture with geometry-based feature projections.

4. Implementation Details

In this section, we discuss the training hyper-parameter choices and schedules used for all the baselines and our TransFuser. We perform single-GPU training for all models on GTX 1080Ti GPU. We train the LBC model using the codebase provided by the authors, which uses the Adam [9] optimizer with a learning rate of 10^{-4} and weight decay of 0.0. The model is validated after every epoch, using a separate partition of our dataset, and the learning rate is reduced by a factor of 2 if the validation loss does not improve for 5 epochs. We train the model for a maximum of 100 epochs and terminate the training if the learning rate goes below 10^{-6} or if the validation loss has not improved for 15 epochs. For all the other models, we use the AdamW optimizer [13], which is a variant of Adam. The learning rate is set to 10^{-4}, weight decay to 0.01, and Adam beta values to the PyTorch defaults of 0.9 and 0.999. All the models are trained for a maximum of 100 epochs. The model is validated after every 5 epochs and we use early stopping to terminate the training if the validation loss has not improved for 25 epochs. We use the checkpoint with the best validation loss for the online evaluation of all the models.

5. Experiments

In this section, we describe the official CARLA Leaderboard metrics[7] in detail and present the results of our submission to the CARLA Leaderboard. We then provide additional results and analysis to supplement our findings in the main paper.

5.1. Metrics

(1) **Route Completion (RC)**: percentage of route distance completed, R_i by the agent in route i, averaged across N routes.

$$\text{RC} = \frac{1}{N}\sum_{i}^{N} R_i. \tag{3}$$

(2) **Infraction Multiplier (IM)**: geometric series of infraction penalty coefficients, p_j for every instance of infraction j incurred by the agent during the route. Agents start with an ideal 1.0 base score, which is reduced by a penalty coefficient for every infraction.

$$\text{IM} = \prod_{j}^{\text{ped,....,stop}} (p_j)^{\#\text{ infractions}_j}. \tag{4}$$

The penalty coefficient for each infraction is pre-defined and set to 0.50 for collision with a pedestrian, 0.60 for collision with a vehicle, 0.65 for collision with static layout, 0.7 for red light violation, and 0.8 for stop sign violation. Besides these, if an agent drives off-road then the route completion is reduced by a multiplier (1- % off route).

(3) **Driving Score (DS)**: weighted average of the route completion with infraction multiplier P_i

$$\text{DS} = \frac{1}{N}\sum_{i}^{N} R_i P_i. \tag{5}$$

(4) **Infractions per km**: the infractions considered are collisions with pedestrians, vehicles, and static elements, running a red light, running a stop sign, off-road infractions, route deviations, timeouts, and vehicle blocked.

[7] https://leaderboard.carla.org

Method	Auxiliary Supervision	Sensors	DS	RC	IM
CILRS [5]	Velocity	1 camera	5.37	14.40	0.55
LBC [2]	BEV Semantics	3 cameras	8.94	17.54	0.73
AIM	None	1 camera	12.88	41.52	0.59
Late Fusion	None	1 camera + LiDAR	13.27	42.10	0.54
Geometric Fusion	None	1 camera + LiDAR	14.47	40.99	0.48
TransFuser (Ours)	None	1 camera + LiDAR	16.93	51.82	0.42
CIL-WP	2D Semantics + Depth	1 camera	19.38	67.02	0.39
NEAT	BEV Semantics	3 cameras	21.83	41.71	0.65
MaRLn [18]	2D Semantics + Affordances	1 camera	24.98	46.97	0.52

Table 2: **CARLA Leaderboard Evaluation.** We report the mean route completion and driving score over 100 secret evaluation routes on the official evaluation server, comparing to the publicly visible submissions.

5.2. CARLA Leaderboard Evaluation

We submit all the models from our study to the CARLA Autonomous Driving Leaderboard which contains a secret set of 100 evaluation routes and report the results in Tab. 2. Among the models that do not use auxiliary supervision, TransFuser achieves the highest DS, outperforming Geometric Fusion by 17%. CIL-WP has a similar architecture to AIM and uses 2D semantics and depth as auxiliary supervision. NEAT employs an implicit function to map locations in BEV space to input image features using attention along with BEV semantic prediction as an auxiliary task. Incorporating auxiliary supervision as in these approaches has shown success in recent work on Imitation Learning, and is likely to also improve the performance of TransFuser [1, 10].

The top Leaderboard entry, MaRLn, builds on top of the Reinforcement Learning (RL) method presented in [18]. In this approach, an encoder is first trained to predict both the 2D semantics and specific affordances such as the scene traffic light state, and the relative position and orientation between the vehicle and lane. The encoder is then frozen and used to train a value function-based RL method. The authors of MaRLn indicated that training a model on a single town required 20 days of simulation time [18]. In comparison, our training dataset for all CARLA towns only requires 21 hours of simulation time and has the potential for further improvements through orthogonal techniques such as Active Learning [3, 4, 7] and DAgger [15, 16]. Further, it is possible to perform RL-based fine-tuning of imitation learning policies like ours, which is computationally expensive but known to be beneficial for performance [14].

5.3. Additional Results

In Table 1(a) of the main paper, we report the mean and standard deviation over 9 evaluation runs of each method in Town05 Short and Town05 Long evaluation settings consisting of scenarios. In Tab. 3, we also provide results on Town05 Short and Town05 Long evaluation settings without any scenarios. We observe that in the absence of scenarios, Late Fusion performs slightly better compared to TransFuser but in the presence of scenarios, TransFuser achieves a higher driving score.

We also provide detailed results (Tab. 4) for 1 run of each model on all the metrics described in Sec. 5.1 on Town05 Long evaluation setting. We observe that our TransFuser model consistently performs well among all the models and across all the metrics. Compared to the Geometric Fusion approach, TransFuser achieves a better DS while sustaining fewer infractions, as evident by significantly higher IM. Specifically, it outperforms Geometric Fusion by 37% on DS and 24% on IM in the No Scenarios setting and by 22% on DS and 78% on IM in the Scenarios setting.

5.4. Attention Map Visualizations

The transformer takes in 64 image feature tokens and 64 LiDAR feature tokens as input where each token corresponds to a 32×32 patch in the input modality. We consider 1000 frames from Town05 intersections and examine the top-5 attention weights for source tokens in image and LiDAR input. We provide visualization of 16 such frames in Fig. 3. We observe that TransFuser is able to map (1) traffic lights in the image input to vehicles in LiDAR BEV, (2) vehicles in LiDAR BEV to same vehicles in image input, (3) vehicles in LiDAR BEV to different vehicles and traffic lights in image input. While TransFuser attends to the objects of interest at intersections, we also observe cases where the dependency between the source entity and the attended entity is inaccurate, e.g. traffic light attends to vehicles in the wrong direction.

Method	**Town05 Short**				**Town05 Long**			
	No Scenarios		Scenarios		No Scenarios		Scenarios	
	DS ↑	RC ↑	DS ↑	RC ↑	DS ↑	RC ↑	DS ↑	RC ↑
CILRS [5]	9.09 ± 2.07	14.09 ± 0.10	7.47 ± 2.51	13.40 ± 1.09	4.75 ± 3.68	8.34 ± 3.55	3.68 ± 2.16	7.19 ± 2.95
LBC [2]	47.95 ± 6.54	72.59 ± 5.00	30.97 ± 4.17	55.01 ± 5.14	11.98 ± 2.70	58.25 ± 9.00	7.05 ± 2.13	32.09 ± 7.40
AIM	64.46 ± 11.34	82.31 ± 13.73	49.00 ± 6.83	81.07 ± 15.59	33.20 ± 6.54	73.77 ± 12.20	26.50 ± 4.82	60.66 ± 7.66
LF	**76.27** ± 9.00	87.12 ± 7.78	51.56 ± 5.24	83.66 ± 11.04	**44.87** ± 5.39	**89.52** ± 9.55	31.30 ± 5.53	68.05 ± 5.39
GF	67.36 ± 7.38	**91.80** ± 6.72	54.32 ± 4.85	**86.91** ± 10.85	30.92 ± 4.02	78.06 ± 10.10	25.30 ± 4.08	**69.17** ± 11.07
TF (Ours)	74.64 ± 3.69	87.63 ± 4.16	**54.52** ± 4.29	78.41 ± 3.75	43.68 ± 2.40	79.64 ± 8.51	**33.15** ± 4.04	56.36 ± 7.14
Expert	*99.86 ± 0.02*	*99.86 ± 0.02*	*84.67 ± 6.21*	*98.59 ± 2.17*	*72.45 ± 8.48*	*100.00 ± 0.00*	*38.60 ± 4.00*	*77.47 ± 1.86*

Table 3: **Driving Performance.** We report the mean and standard deviation over 9 runs of each method (3 training seeds, each trained model evaluated 3 times) on Route Completion (RC) and Driving Score (DS) in Town05 Short and Town05 Long settings with scenario and without scenarios. LF: Late Fusion, GF: Geometric Fusion, TF: TransFuser

Method	DS ↑	RC ↑	IM ↑	Ped ↓	Veh ↓	Static ↓	Red ↓	Stop ↓	Off-road ↓	Dev ↓	TO ↓	Blocked ↓
	Town05 Long No Scenarios											
CILRS [5]	6.96	10.96	0.64	0	2.26	1.97	0.06	0	1.39	0.67	0	2.09
LBC [2]	9.16	56.52	0.21	0	0.17	0.42	0.52	0.03	0.10	0.02	0.02	0.20
AIM	30.24	65.19	0.49	0	0.10	0.66	0.06	0.03	0.40	0	0.01	0.45
Late Fusion	39.49	85.75	0.46	0	0.04	0	0.10	0.15	0.02	0	0.01	0.05
Geometric Fusion	33.32	86.38	0.45	0	0.03	0	0.17	0.12	0.07	0	0	0.05
TransFuser (Ours)	41.69	82.75	0.59	0	0.03	0	0.09	0.03	0.02	0	0.01	0.25
	Town05 Long Scenarios											
CILRS [5]	4.84	9.66	0.65	0	0.33	2.07	0.15	0	2.11	0.65	0	2.6
LBC [2]	6.17	24.59	0.48	0	0.23	0.32	0.37	0	0.15	0	0	1.35
AIM	30.83	52.96	0.71	0	0.08	0.02	0.01	0.02	0.02	0	0	0.52
Late Fusion	34.96	60.33	0.70	0.02	0.06	0.02	0.03	0.01	0.12	0	0.01	0.46
Geometric Fusion	29.17	81.09	0.37	0.02	0.18	0.17	0.07	0.03	0.06	0	0.03	0.19
TransFuser (Ours)	39.66	66.19	0.63	0.01	0.06	0	0.14	0.02	0.11	0	0	0.17

Table 4: **Detailed Results.** We report detailed results on all the metrics for 1 run of each model on the Town05 Long evaluation setting. DS: Driving Score, RC: Route Completion, IM: Infraction Multiplier, Ped: Collision with pedestrian, Veh: Collision with vehicle, Static: Collision with static layout, Red: Red light violation, Stop: Stop sign infractions, Off-road: Off-road driving, Dev: Route deviation, TO: Timeout, Blocked: Vehicle Blocked

5.5. Ablation on Model Capacity

All the fusion methods in the main paper use ResNet-34 as the image encoder and ResNet-18 as the LiDAR BEV encoder. In our preliminary experiments, we experimented with using different perception backbone for the Late Fusion model to investigate if increasing the model capacity or changing the architecture of the perception backbone affects the performance of the model. We observed a relative decrease of (1) 20.14% when using ResNet34 for LiDAR (2) 30.77% when using Inception v3 for image (3) 52.83% when using Inception v3 for both image and LiDAR. This indicates that simply adding more layers and increasing the depth of the perception backbone hampers performance, potentially due to overfitting.

References

[1] Aseem Behl, Kashyap Chitta, Aditya Prakash, Eshed Ohn-Bar, and Andreas Geiger. Label efficient visual abstractions for autonomous driving. In *Proc. IEEE International Conf. on Intelligent Robots and Systems (IROS)*, 2020. 8

[2] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Proc. Conf. on Robot Learning (CoRL)*, 2019. 2, 3, 4, 8, 9

[3] Kashyap Chitta, Jose M. Alvarez, Elmar Haussmann, and Clement Farabet. Training data subset search with ensemble active learning. *arXiv.org*, 1905.12737, 2019. 8

[4] Kashyap Chitta, Jose M. Alvarez, and Adam Lesnikowski. Large-scale visual active learning with deep probabilistic ensembles. *arXiv.org*, 1811.03575, 2018. 8

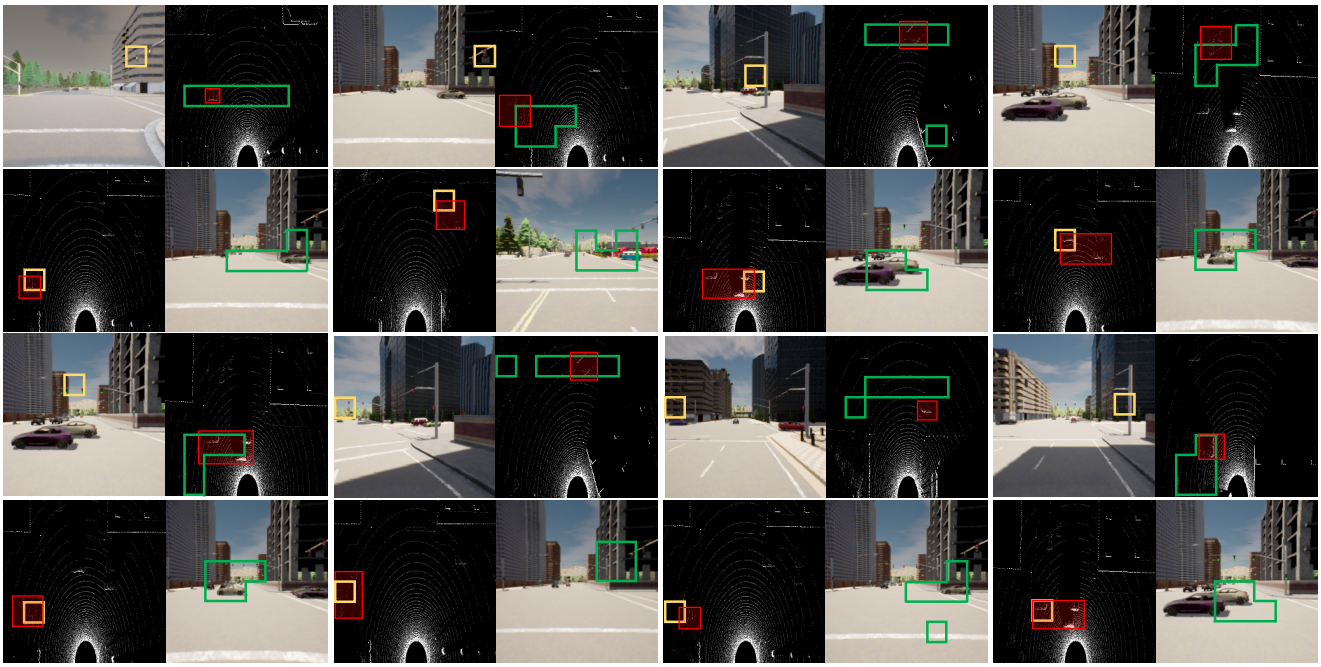


Figure 3: **Attention Maps.** For the yellow query token, we show the top-5 attended tokens in green and highlight the presence of vehicles in the LiDAR in red. TransFuser attends to the vehicles and traffic lights at intersections in the input modalities, albeit at a slightly different location.

[5] Felipe Codevilla, Eder Santana, Antonio M. López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019. 4, 8, 9

[6] Jia Deng, Wei Dong, Richard Socher, Li jia Li, Kai Li, and Li Fei-fei. Imagenet: A large-scale hierarchical image database. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2009. 5

[7] Elmar Haussmann, Michele Fenzi, Kashyap Chitta, Jan Ivanecky, Hanson Xu, Donna Roy, Akshita Mittel, Nicolas Koumchatzky, Clement Farabet, and Jose M. Alvarez. Scalable Active Learning for Object Detection. In *Proc. IEEE Intelligent Vehicles Symposium (IV)*, 2020. 8

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2015. 7

[10] Zhihao Li, Toshiyuki Motoyoshi, Kazuma Sasaki, Tetsuya Ogata, and Shigeki Sugano. Rethinking self-driving: Multi-task knowledge for better generalization and accident explanation ability. *arXiv.org*, abs/1809.11100, 2018. 8

[11] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 6

[12] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2018. 6

[13] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2019. 7

[14] Eshed Ohn-Bar, Aditya Prakash, Aseem Behl, Kashyap Chitta, and Andreas Geiger. Learning situational driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 8

[15] Aditya Prakash, Aseem Behl, Eshed Ohn-Bar, Kashyap Chitta, and Andreas Geiger. Exploring data aggregation in policy learning for vision-based urban autonomous driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 8

[16] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011. 8

[17] Ibrahim Sobh, Loay Amin, Sherif Abdelkarim, Khaled Elmadawy, M. Saeed, Omar Abdeltawab, M. Gamal, and Ahmad El Sallab. End-to-end multi-modal sensors fusion system for urban automated driving. In *Advances in Neural Information Processing Systems (NeurIPS) Workshops*, 2018. 6

[18] Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 8